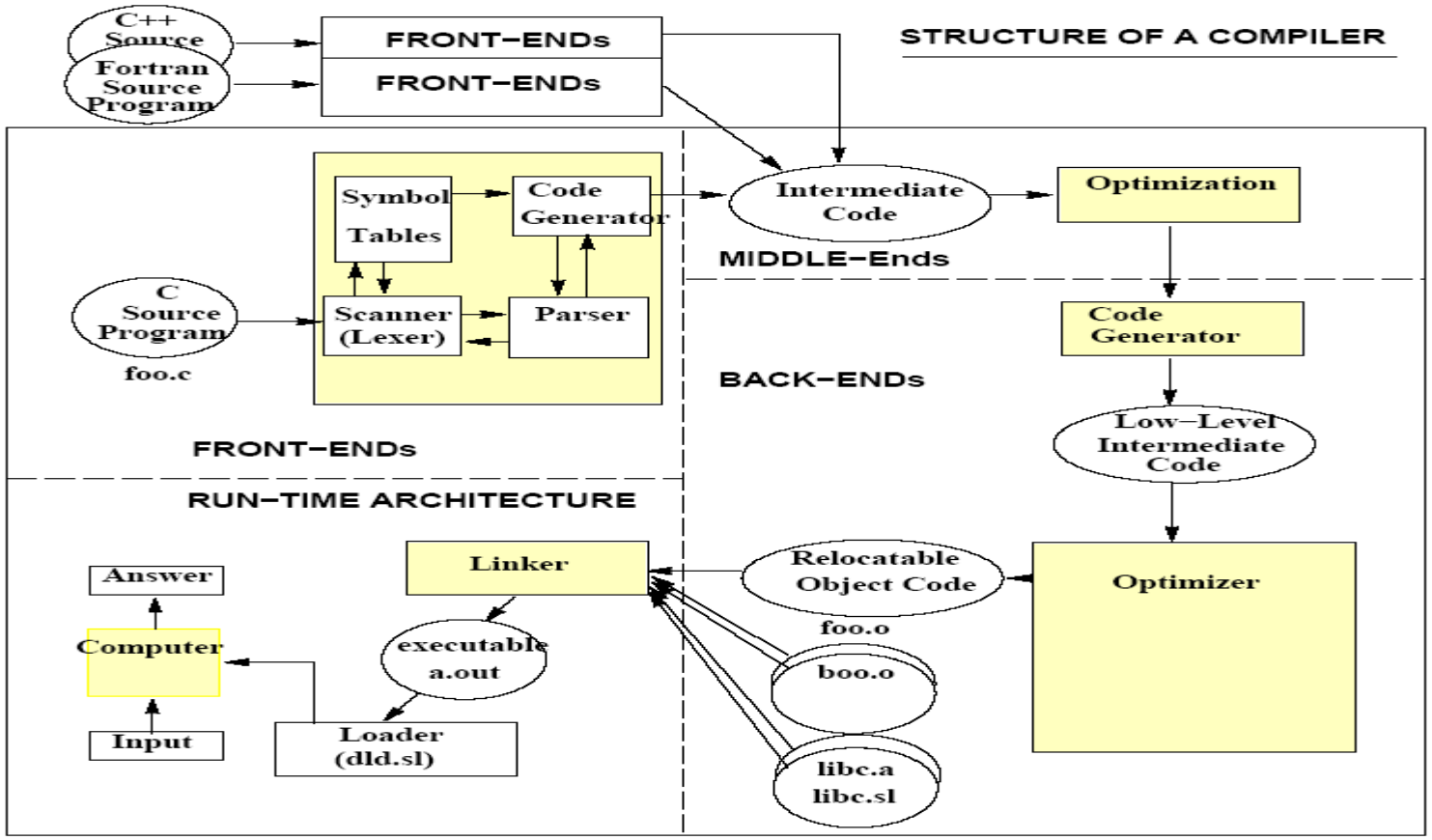
Introduction

- What is the topic of this course?
- What can optimizing compilers do?
- What do you get out of this class?

Topic of this Course

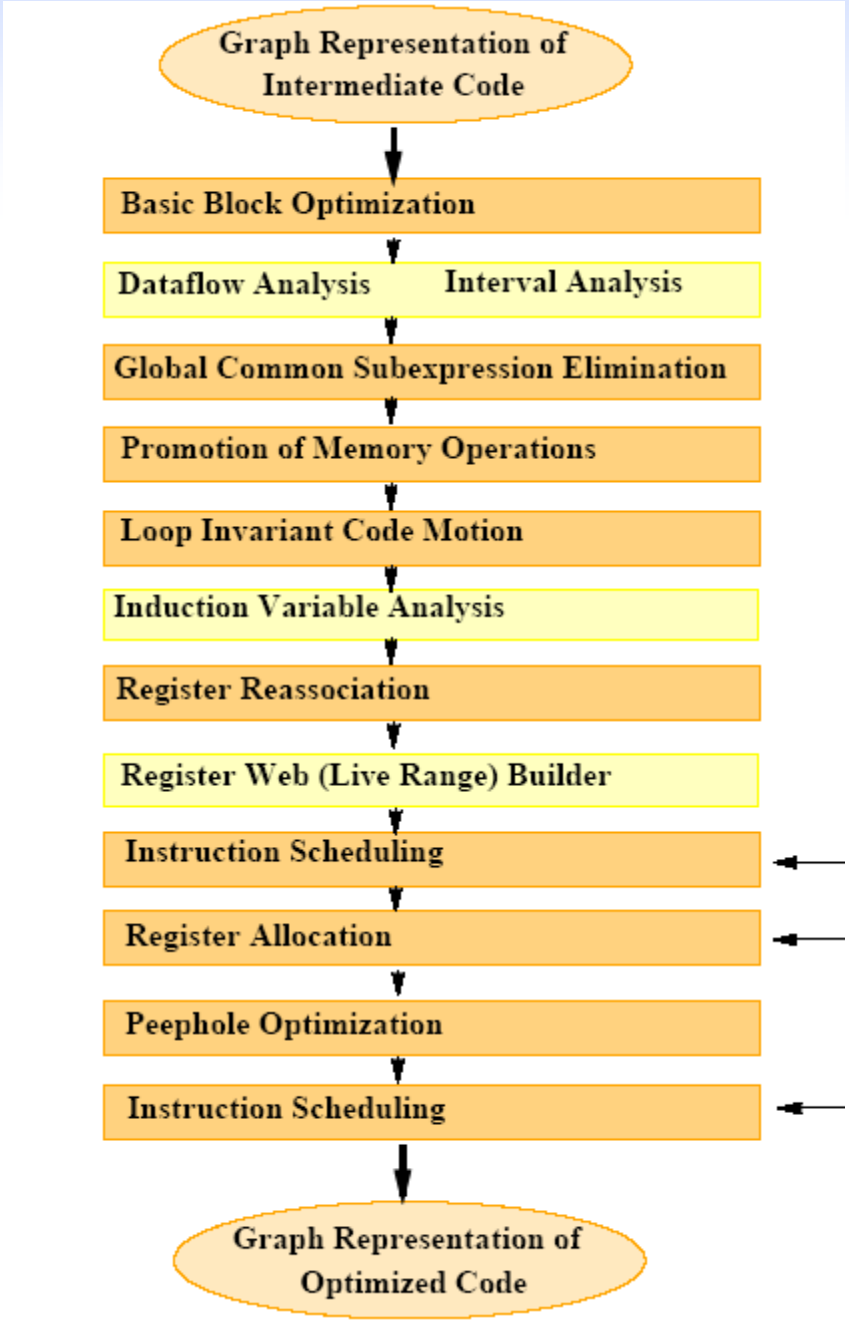
- In your undergraduate compiler class, you learned about compiler front-ends
  - Lexical analysis, syntax analysis, symbol table, semantic analysis, intermediate code generation
- Now, we will talk about compiler back-ends
  - Especially, focus on compiler optimizations
  - Also deal with run-time architectures briefly

Structure of Modern Compilers



What are Compiler Optimizations?

- Optimization
  - Transform a computation to an equivalent but better computation
  - Not actually optimal
- Optimization is done via phases
  - Code becomes better as passing thru phases
  - Phase ordering issues


Graph Representation of Intermediate Code
Basic Block Optimization
Dataflow Analysis
Interval Analysis
Global Common Subexpression Elimination
Promotion of Memory Operations
Loop Invariant Code Motion
Induction Variable Analysis
Register Reassociation
Register Web (Live Range) Builder
Instruction Scheduling
Register Allocation
Peephole Optimization
Instruction Scheduling
Graph Representation of Optimized Code

How Optimizations Affect Performance

- Execution Time of a Program:
  1. Instruction Count (# of Instructions Executed)
  2. CPI (Average # of Cycles/Instruction)
  3. Cycle Time of the Machine

- Compiler optimizations affect (1) and (2)

What Compiler Optimizations Can Do

- Reduce the number of executed instructions
- Replace expensive instructions with simpler ones
  - e.g., replace multiplication with addition or shift
- Reduce cache misses
  - Both instruction cache and data cache
- Group independent instructions for parallel execution
  - Superscalar or EPIC (VLIW) microprocessors

Sometimes code size is more important than the speed

- Embedded CPUs: microcontroller or DSPs

Why Compilers are Interesting?

- For improving overall program performance
  - Speedup of system = hardware + compiler
  - Compiler optimization is important in performance report
    - e.g., refer to SPEC web pages (www.spec.org)
- Compilers affect CPU architecture designs
  - E.g., prefetch instructions, address compare buffers, EPIC, speculative loads, multi-threading, single-chip MP, ...

Why Compilers are Interesting?

- Compiler optimization is a key technology to virtual machine (VM), which is a software for cross-platform (ISA or OS) compatibility
  - A platform can run programs bounded to a different platform using a VM
  - VM translates one ISA to another ISA and execute it, which will be much slower than native execution
  - Code optimizations can reduce the performance gap
  - VM with dynamic code optimization can even improve native execution speed, or for co-designed ISA (e.g., Transmeta Crusoe)

Why Compilers are Interesting?

- An example of a large software system
  - Debugging complexity is extremely high
    - Why? The output of a compiler is code, not data
  - If you are working on a H/W, please never ask
    - "*Make a compiler for my chip within a month*"
  - There are many software engineering Issues

Optimizations Affect the ISA Design

- An Example that can exploit Store/Load-base update instructions in IBM Power or HP PA-RISC

```
#define N 100
main ()
{
   int A[N], i;
   for (i = 0; i < N; i++) A[i] = 0;
}
```

Optimized Assembly Code

```
    LDO     -440(%r30),%r31
    LDI     -100,%r23
$003
        ADDIB,< 1,%r23,$003
        STWS,MA %r0,4(%r31)  ; 0 -> A[i], i ++.
$002    -------------------  ---------------
```

- There are only TWO instructions in the loop!
- Update instructions assume this type of optimization

Problem Solving in Optimizing Compilers

- Find common cases
- Formulate mathematically
- Decide appropriate phase(s)
- Develop algorithms
- Implement
- Evaluate on real data

Never work on optimization that has little cases

What do You Get out of this Course?

- Basic knowledge of modern optimizing compilers
  - Basic program analysis and code transformation techniques
  - Some mathematical & graph theory for optimizations
  - Learn some advanced optimizations techniques
- Personally, I believe optimization techniques are hard to understand unless you actually implement & evaluate them
- On the other hand, it is very hard to implement/debug and evaluate them in a short period of time
- If you want some real knowledge, try to implement some or read some of the open-source compiler code (SUIF, ORC, ...)

What Can Optimizations Do for You?

- Let's See an Example: Bubble Sort Program

```
#define N 100
main ()
{
        int A[N], i, j;
        int temp;
        for (i = N-1; i >= 0; i--)
            for (j = 0; j < i; j++)
            {
               if (A[j] > A[j+1]) {
                    temp = A[j];
                    A[j] = A[j+1];
                    A[j+1] = temp;
                }
             }
}
```

- We Compiled With/Without Optimizations for the PA-RISC (i.e., cc –S test.c and cc –O –S test.c, repectively)

```
 LDI      99,%r1                          |    LDWX,S  %r20(%r21),%r22; A[j+1]
 STW      %r1,-48(%r30)   ; 99->i         |    LDW     -44(%r30),%r1   ; j
 LDW      -48(%r30),%r31                  |    LDO     -448(%r30),%r31; &A
 COMIBF,<=,N 0,%r31,$002; i>=0 ?          |    SH2ADD  %r1,%r31,%r19   ; A[j]
$003                                      |    STWS    %r22,0(%r19);A[j+1]->A[j]
 STW      %r0,-44(%r30)   ; 0->j          |    LDW     -44(%r30),%r20;
 LDW      -44(%r30),%r19                  |    LDO     1(%r20),%r21
 LDW      -48(%r30),%r20                  |    LDW     -40(%r30),%r22
 COMBF,<,N %r19,%r20,$001;j<i ?           |    LDO     -448(%r30),%r1
$006                                      |    SH2ADD  %r21,%r1,%r31
 LDW      -44(%r30),%r21                  |    STWS    %r22,0(%r31);temp->A[j+1]
 LDO      1(%r21),%r22    ; j+1           |$004
 LDO      -448(%r30),%r1 ; &A             |    LDW     -44(%r30),%r19 ; j
 LDW      -44(%r30),%r31 ; j              |    LDO     1(%r19),%r20    ; j++
 LDWX,S   %r31(%r1),%r19 ; A[j]           |    STW     %r20,-44(%r30)
 LDO      -448(%r30),%r20; &A             |    LDW     -44(%r30),%r21
 LDWX,S   %r22(%r20),%r21; A[j+1]         |    LDW     -48(%r30),%r22 ; i
 COMB,<=,N %r19,%r21,$004;A[j]<A[j+1]     |    COMB,<  %r21,%r22,$006 ; j<i ?
 LDO      -448(%r30),%r22 ;&A             |    NOP
 LDW      -44(%r30),%r1    ; j            |$001
 LDWX,S   %r1(%r22),%r31   ; A[j]         |    LDW     -48(%r30),%r1   ; i
 STW      %r31,-40(%r30) ;A[j]->temp      |    LDO     -1(%r1),%r31    ; i--
 LDW      -44(%r30),%r19                  |    STW     %r31,-48(%r30) ;
 LDO      1(%r19),%r20     ; j+1          |    LDW     -48(%r30),%r19
 LDO      -448(%r30),%r21 ; &A            |    COMIB,<=  0,%r19,$003 ; i>=0 ?
```

Optimized Assembly Code

```
    LDI     99,%r31
$003
    COMBF,<,N       %r0,%r31,$001
    LDO     -444(%r30),%r23
    SUBI    0,%r31,%r24
$006
    LDWS    -4(%r23),%r25
    LDWS,MA 4(%r23),%r26
    COMB,<=,N       %r25,%r26,$007
    STWS    %r26,-8(%r23)
    STWS    %r25,-4(%r23)
$007
    ADDIB,<,N       1,%r24,$006+4
    LDWS    -4(%r23),%r25
$001
    ADDIBF,<        -1,%r31,$003
    NOP
$002
```

- Compare the number of instructions in the Loop!